National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Chethan Sureshbabu
StudentID:x21235091

School of Computing
National College of Ireland

Supervisor: Mrs Harshani Nagahamulla

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Chethan Sureshbabu
Student ID:	x21235091
Programme:	MSc in Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Mrs Harshani Nagahamulla
Submission Due Date:	31/01/2024
Project Title:	Configuration Manual
Word Count:	928
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Chethan Sureshbabu
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Chethan Sureshbabu
x21235091

1 Introduction

The configuration manual serves as an essential guide, providing a detailed walkthrough for the proper execution of the Multimodal Stress Analysis project. It encompasses step-by-step procedures and crucial information about system requirements, software specifications, and library versions necessary for the successful implementation of the project. Following this manual meticulously ensures a smooth setup and execution, allowing users to delve into the intricate process of analyzing stress across multiple modalities seamlessly.

2 Hardware Specifications

In this section, the hardware specifications of the local machine are described.

2.1 Hardware Specification of the Local Machine

Item	Value
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-STP96E14
System Manufacturer	Acer
System Model	Aspire A715-41G
System Type	x64-based PC
System SKU	0000000000000000
Processor	AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx, 2100 Mhz, 4 Core(s), 8
BIOS Version/Date	INSYDE Corp. V1.01, 05-05-2020
SMBIOS Version	3.1
Embedded Controller Version	1.01
BIOS Mode	UEFI
BaseBoard Manufacturer	PK
BaseBoard Product	Azalea_PKS
BaseBoard Version	V1.01
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.22621.2506"
User Name	LAPTOP-STP96E14\chethan
Time Zone	GMT Standard Time
Installed Physical Memory (RA...	8.00 GB
Total Physical Memory	5.94 GB
Available Physical Memory	830 MB
Total Virtual Memory	21.9 GB
Available Virtual Memory	11.5 GB
Page File Space	16.0 GB
Page File	C:\pagefile.sys

Figure 1: System Summary

The above Figure 1 details the Hardware specifications of the local machine to implement or execute the project.

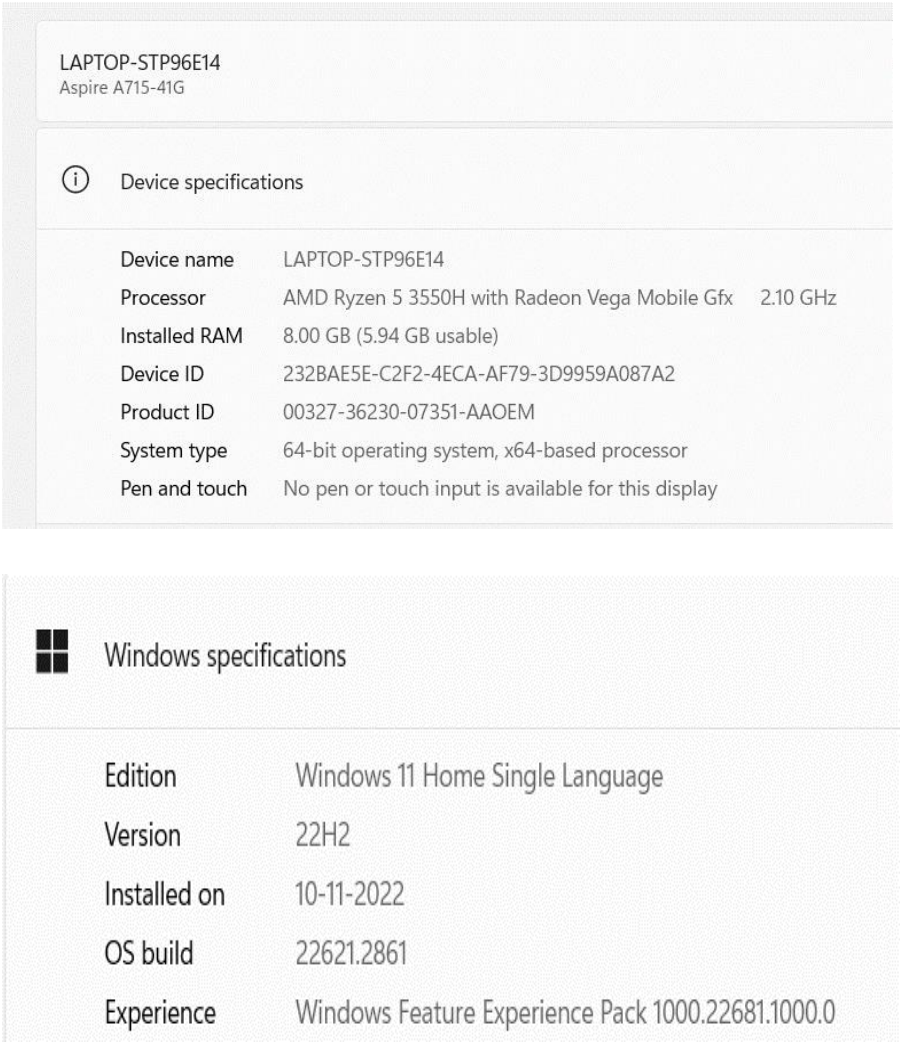

Figure 2: Hardware and Device Information

The above Figure 2 shows describes the device specification and Windows specification of the local machine.

3 Software tools required to run/execute the project.

The applications used to implement or complete the project are mentioned below:

- Anaconda navigator 2.5.0
- Jupiter Notebook version 6.5.4

- Python was used as a programming language.
- TensorFlow 2.14.0
- Sklearn 1.3.2

4 Python Libraries were used.

```python
#importing libraries for text model
import re
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.stem import WordNetLemmatizer
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import time
from sklearn.metrics import roc_curve, auc
from joblib import dump
```

Figure 3: Importing libraries for text model.

```python
#importing libraries for audio model
import os
import librosa
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from sklearn.preprocessing import StandardScaler, LabelEncoder
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
from joblib import dump
```

Figure 3: Importing libraries for an audio model.

```python
#importing libraries for image model
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import EfficientNetB0
from efficientnet.keras import center_crop_and_resize
import seaborn as sns
import matplotlib.pyplot as plt
from joblib import dump
```

Figure 4: Importing libraries for an image model.

```
#importing libraries for late fusion model
import numpy as np
from joblib import load, dump
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.calibration import calibration_curve
from sklearn.metrics import log_loss
from sklearn.metrics import brier_score_loss
```

Figure 5: Importing libraries for a Late Fused model.

The above figure shows the description of the Python libraries we have used while implementing Text, Audio, Image, and Late Fusion models.

6 Datasets used in the project.

Dataset 1 - The sentiment140 dataset is used for the Text Model, It contains 1,600,000 tweets extracted using the Twitter API.

Dataset 2 - Facial Expressions Training Data is used for the Image Model implementation, This data set is based on Affect Net-HQ, Affect Net is a large database of faces labeled by "affects"

Dataset 3 - RAVDESS Emotional speech audio data set is used for implementing the Audio model. This portion of the RAVDESS contains 1440 files The RAVDESS contains 24 professional actors (12 female, 12 male), vocalizing two lexically matched statements in a neutral North American accent. Speech emotions include calm, happy, sad, angry, fearful, surprised, and disgusted expressions.

So, in this section, we have discussed the three data sets we have used for the Multimodal Stress Analysis project.

7 Code Snippets

In this section, important code snippets from the Text, Audio, Image, and Late Fusion models we have implemented will be discussed.

7.1 Text Model Implementation

```
# Display unique values in the 'sentiment' column
unique_sentiments = dataset['sentiment'].unique()
print("Unique Sentiments:", unique_sentiments)

# Adjust the mapping based on your sentiment values
sentiment_mapping = {0: 'high stressed', 4: 'low stressed'}
dataset['stress_level'] = dataset['sentiment'].map(sentiment_mapping)

# Checking the class counts
class_counts = dataset['stress_level'].value_counts()

print("Class Counts:")
print(class_counts)
```

```
Unique Sentiments: [0 4]
Class Counts:
stress_level
low stressed     800000
high stressed    799999
Name: count, dtype: int64
```

```
# Storing data in lists
text, stress_levels = list(dataset['text']), list(dataset['stress_level'])

```

```
dataset.head()
```

	sentiment	ids	date	flag	user	text	stress_level
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...	high stressed
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...	high stressed
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire	high stressed
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....	high stressed
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew	high stressed

Fig 6: Represents the Mapping function.

After loading the text data set we applied the mapping function to the data to add the stress levels column in the dataset.

```
import time
t = time.time()
processedtext = preprocess(text)
print(f'Text Preprocessing complete.')
print(f'Time Taken: {round(time.time()-t)} seconds')
```

```
Text Preprocessing complete.
Time Taken: 308 seconds
```

```
# Splitting the Data
X_train, X_test, y_train, y_test = train_test_split(processedtext, stress_levels, test_size=0.05, random_state=0)

```

```
# Create the folder if it doesn't exist
save_path = 'C:\\Users\\chethan\\OneDrive\\Desktop\\Text Test Data'
os.makedirs(save_path, exist_ok=True)
```

```
# Combine the text and labels for the selected test samples
selected_test_data = list(zip(X_test, y_test))
```

```
# TF-IDF Vectorizer
vectoriser = TfidfVectorizer(ngram_range=(1, 2), max_features=500000)
vectoriser.fit(X_train)
```

```
# Transforming the data
X_train = vectoriser.transform(X_train)
X_test = vectoriser.transform(X_test)
```

```
# Evaluate Model Function
def model_Evaluate(model):
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
    cf_matrix = confusion_matrix(y_test, y_pred)
```

Figure 7: Important preprocessing steps

In figure 7 we can see the most important preprocessing steps we have we have done before building the model.

```
# Logistic Regression Model
LRmodel = LogisticRegression(C=2, max_iter=1000, n_jobs=-1)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)
```

```
                precision    recall  f1-score   support

 high stressed       0.83      0.82      0.83     39986
  low stressed       0.82      0.83      0.83     40014

      accuracy                           0.83     80000
     macro avg       0.83      0.83      0.83     80000
  weighted avg       0.83      0.83      0.83     80000
```

Figure 8: Model building

In the above code snippet we can see the code for implementing the Logistic Regression model and also the classification report of the model.

```
from joblib import dump

# Replace 'C:/Users/chethan/Downloads/RIC/logistic_regression_model.joblib' with your desired path and filename
save_path_lr = 'C:/Users/chethan/Downloads/RIC/logistic_regression_model.joblib'

# Save the Logistic Regression model
dump(LRmodel, save_path_lr)

print(f"Logistic Regression model saved successfully at: {save_path_lr}")
```

```
Logistic Regression model saved successfully at: C:/Users/chethan/Downloads/RIC/logistic_regression_model.joblib
```

Figure 9: Saving the Logistic Regression model

Here, we have saved the model to use in the future for the Late Fusion Model

7.2 Audio Model Implementation

```
# Function to load RAVDESS dataset
def load_ravdess_dataset(dataset_path):
    data = {'path': [], 'emotion': []}

    # Iterate over actor folders
    for actor_folder in os.listdir(dataset_path):
        actor_path = os.path.join(dataset_path, actor_folder)

        if os.path.isdir(actor_path):
            # Iterate over audio files in each actor folder
            for audio_file in os.listdir(actor_path):
                audio_path = os.path.join(actor_path, audio_file)

                # Extract emotion from the file name (assuming file names are structured)
                emotion = int(audio_file.split('-')[2])

                # Append data to the dictionary
                data['path'].append(audio_path)
                data['emotion'].append(emotion)

    # Create a DataFrame from the dictionary
    df = pd.DataFrame(data)

    return df

# path to the dataset
dataset_path = r'C:\\Users\\chethan\\Downloads\\RIC\\audio_speech_actors_01-24'

# Loading the dataset
ravdess_dataset = load_ravdess_dataset(dataset_path)
```

Figure 10: Function to load the Audio dataset.

Using the above function we have successfully loaded the Audio dataset.

```
# Mapping emotion labels to stress levels
stress_mapping = {
    1: 'low_stress',
    2: 'low_stress',
    3: 'low_stress',
    4: 'low_stress',
    5: 'high_stress',
    6: 'high_stress',
    7: 'high_stress',
    8: 'high_stress',
}

ravdess_dataset['stress_level'] = ravdess_dataset['emotion'].map(stress_mapping)

```

```
# Displaying the loaded dataset
print(ravdess_dataset.head())
```

```
                                                path  emotion stress_level
0  C:\\Users\\chethan\\Downloads\\RIC\\audio_spee...        1   low_stress
1  C:\\Users\\chethan\\Downloads\\RIC\\audio_spee...        1   low_stress
2  C:\\Users\\chethan\\Downloads\\RIC\\audio_spee...        1   low_stress
3  C:\\Users\\chethan\\Downloads\\RIC\\audio_spee...        1   low_stress
4  C:\\Users\\chethan\\Downloads\\RIC\\audio_spee...        2   low_stress
```

Figure 11: Mapping the stress levels.

Then using the above function we have mapped the stress levels in the dataset.

```
# Function to extract features from audio files
def extract_features(file_path):
    try:
        # Loading audio file with librosa
        audio, sample_rate = librosa.load(file_path, res_type='kaiser_fast')

        # Extracting features using librosa
        mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=13), axis=1)
        chroma = np.mean(librosa.feature.chroma_stft(y=audio, sr=sample_rate), axis=1)
        mel = np.mean(librosa.feature.melspectrogram(y=audio, sr=sample_rate), axis=1)
        contrast = np.mean(librosa.feature.spectral_contrast(y=audio, sr=sample_rate), axis=1)
        tonnetz = np.mean(librosa.feature.tonnetz(y=audio, sr=sample_rate), axis=1)

        # Combining all features into a single array
        feature_array = np.concatenate((mfccs, chroma, mel, contrast, tonnetz))

    except Exception as e:
        print(f"Error encountered while parsing file: {file_path}. Error: {e}")
        return None

    return feature_array
```

```
# Extracting features from each audio file and create a DataFrame
!pip install resampy
ravdess_dataset['features'] = ravdess_dataset['path'].apply(extract_features)

```

Figure 12: Function to extract the features.

Using the above function we can extract the audio features from the audio data set.

```
# Converting features to a list and labels to a NumPy array
X = np.array(ravdess_dataset['features'].tolist())
y = np.array(ravdess_dataset['stress_level'])
```

```
# Encoding the target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_categorical = to_categorical(y_encoded)
```

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42)

```

```
# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 13: The important preprocessing steps before implementing the model.

```
# Building the neural network model
model = Sequential()
model.add(Dense(256, input_shape=(X_train_scaled.shape[1],), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
```

```
# Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

```

```
# Training the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)
```

Figure 14: CNN Model implementation.

The above code is used for model implementation.

```
# Making predictions on the test set
y_pred = model.predict(X_test_scaled)
```

```
9/9 [==============================] - 0s 5ms/step
```

```
# Convertting predictions to class labels
y_pred_labels = le.inverse_transform(np.argmax(y_pred, axis=1))
y_test_labels = le.inverse_transform(np.argmax(y_test, axis=1))
```

```
# Evaluate the model
print("Classification Report:\n", classification_report(y_test_labels, y_pred_labels))
print("Accuracy:", accuracy_score(y_test_labels, y_pred_labels))
```

```
Classification Report:
              precision    recall  f1-score   support

 high_stress       0.83      0.87      0.85       151
  low_stress       0.85      0.80      0.82       137

    accuracy                           0.84       288
   macro avg       0.84      0.84      0.84       288
weighted avg       0.84      0.84      0.84       288

Accuracy: 0.8368055555555556
```

Figure 15: Model prediction on test data.

```python
from joblib import dump

# Save the audio model
save_path = 'C:/Users/chethan/Downloads/RIC/audio_model_saved.joblib'
dump(model, save_path)

print(f"Audio model saved successfully at: {save_path}")
```

```
Audio model saved successfully at: C:/Users/chethan/Downloads/RIC/audio_model_saved.joblib
```

Figure 16: saving the model.

7.3 Image Model Implementation

```python
path = 'C:\\Users\\chethan\\Downloads\\RIC\\Image dataset\\'
file = os.path.join(path, 'labels.csv')
# Load the dataset
emotion_dataset = pd.read_csv(file)
emotion_dataset.head()
```

	Unnamed: 0	pth	label	relFCs
0	0	anger/image0000006.jpg	surprise	0.873142
1	1	anger/image0000060.jpg	anger	0.852311
2	2	anger/image0000061.jpg	anger	0.800957
3	3	anger/image0000066.jpg	disgust	0.843079
4	4	anger/image0000106.jpg	anger	0.849108

Figure 17: Loading the image dataset

```python
# Map existing labels to stress levels
stress_level_mapping = {
    'anger': 'high_stressed',
    'contempt': 'low_stressed',
    'disgust': 'high_stressed',
    'fear': 'high_stressed',
    'happy': 'low_stressed',
    'neutral': 'low_stressed',
    'sad': 'high_stressed',
    'surprise': 'low_stressed'
}
```

```python
# Add a new column 'stress_level' based on the mapping
emotion_dataset['stress_level'] = emotion_dataset['label'].map(stress_level_mapping)

```

```python
# Adding the 'stress_level' column in your DataFrame
high_stressed_count = emotion_dataset[emotion_dataset['stress_level'] == 'high_stressed'].shape[0]
low_stressed_count = emotion_dataset[emotion_dataset['stress_level'] == 'low_stressed'].shape[0]
print("Number of High Stressed Instances:", high_stressed_count)
print("Number of Low Stressed Instances:", low_stressed_count)

```

```
Number of High Stressed Instances: 13118
Number of Low Stressed Instances: 15057
```

Figure 18: Mapping the stress levels.

```python
# Min-Max Scaling
scaler = MinMaxScaler()
emotion_dataset[['relFCs']] = scaler.fit_transform(emotion_dataset[['relFCs']])

```

```python
# Train-test split
X_train_features, X_test_features, y_train, y_test = train_test_split(
    emotion_dataset[['relFCs', 'pth']], emotion_dataset['stress_level'], test_size=0.2, random_state=42
)
```

```python
# Data augmentation using ImageDataGenerator
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```python
# Reshape the data for RGB images
image_height = 96
image_width = 96
num_channels = 3  # For RGB images
```

```python
# Load and preprocess the images for training with data augmentation
X_train_images = []
for image_path in X_train_features['pth']:
    img = load_img(os.path.join(path, image_path), target_size=(image_height, image_width))
    img_array = img_to_array(img)
    img_array = train_datagen.random_transform(img_array)
    X_train_images.append(img_array)

X_train_images = np.array(X_train_images)
```

```python
# Load and preprocess the images for testing
X_test_images = []
for image_path in X_test_features['pth']:
    img = load_img(os.path.join(path, image_path), target_size=(image_height, image_width))
    img_array = img_to_array(img)
    X_test_images.append(img_array)

X_test_images = np.array(X_test_images)
```

Figure 19: Important preprocessing steps before implementing the model.

```python
# EfficientNetB0 model
def build_efficientnet_model(input_shape=(image_height, image_width, num_channels), num_classes=1):
    base_model = EfficientNetB0(input_shape=input_shape, include_top=False, weights='imagenet')

    model = Sequential()
    model.add(base_model)
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='sigmoid'))
    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

```python
# Initialize the model
model = build_efficientnet_model()
```

```python
# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

```

```python
# Encode the target variable 'stress_level'
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

```python
# Train the model
history = model.fit(X_train_images, y_train_encoded, epochs=10, batch_size=32,
                    validation_split=0.2, callbacks=[early_stopping])

```

```
Epoch 1/10
559/559 [==============================] - 846s 1s/step - loss: 0.6692 - accuracy: 0.6289 - val_loss: 0.5827 - val_accuracy: 0.
7018
Epoch 2/10
559/559 [==============================] - 757s 1s/step - loss: 0.5065 - accuracy: 0.7527 - val_loss: 0.5456 - val_accuracy: 0.
7311
```

Figure 20: EfficientnetBO Model implementation.

```python
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test_images, y_test_encoded)
print(f'Test Accuracy: {test_acc}')

```

```
175/175 [==============================] - 56s 317ms/step - loss: 0.5047 - accuracy: 0.7591
Test Accuracy: 0.7590900659561157
```

```python
# Make predictions
y_pred = model.predict(X_test_images)
y_pred_binary = (y_pred > 0.5).astype(int)
```

```
175/175 [==============================] - 59s 316ms/step
```

```python
# Classification report
print(classification_report(y_test_encoded, y_pred_binary))

```

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74      2607
           1       0.77      0.78      0.77      2976

    accuracy                           0.76      5583
   macro avg       0.76      0.76      0.76      5583
weighted avg       0.76      0.76      0.76      5583
```

Figure 21: Model prediction on test data.

```python
from joblib import dump

save_path_efficientnet = 'C:/Users/chethan/Downloads/RIC/efficientnet_model.joblib'

# Save the EfficientNet model
dump(model, save_path_efficientnet)

print(f"EfficientNet model saved successfully at: {save_path_efficientnet}")
```

```
EfficientNet model saved successfully at: C:/Users/chethan/Downloads/RIC/efficientnet_model.joblib
```

Figure 22: Saving the model for using later for the late fusion model.

7.4 Late Fusion Model Implementation

```python
# Load the saved models
audio_model = load('C:/Users/chethan/Downloads/RIC/audio_model_saved.joblib')
image_model = load('C:/Users/chethan/Downloads/RIC/efficientnet_model.joblib')
text_model = load('C:/Users/chethan/Downloads/RIC/logistic_regression_model.joblib')
tfidf_vectorizer = load('C:/Users/chethan/Downloads/RIC/tfidf_vectorizer.joblib')
```

Figure 23: Loading the Text, Audio, and image Models.

```python
# Load the saved models
audio_model = load('C:/Users/chethan/Downloads/RIC/audio_model_saved.joblib')
image_model = load('C:/Users/chethan/Downloads/RIC/efficientnet_model.joblib')
text_model = load('C:/Users/chethan/Downloads/RIC/logistic_regression_model.joblib')
tfidf_vectorizer = load('C:/Users/chethan/Downloads/RIC/tfidf_vectorizer.joblib')

#  test data for audio, image, and text
audio_test_data = X_test_audio
video_test_data = X_test_images
text_test_data = X_test_text

# Obtain predictions from individual models
audio_predictions = audio_model.predict(audio_test_data)
image_predictions = image_model.predict(video_test_data)
text_predictions_proba = text_model.predict_proba(tfidf_vectorizer.transform(text_test_data))[:, 1]

# Ensuring all predictions have the same number of samples
min_samples = min(len(audio_predictions), len(image_predictions), len(text_predictions_proba))
audio_predictions = audio_predictions[:min_samples]
image_predictions = image_predictions[:min_samples]
text_predictions_proba = text_predictions_proba[:min_samples]

# Concatenate features
all_features = np.concatenate((audio_predictions[:, 0].reshape(-1, 1), image_predictions, text_predictions_proba.reshape(-1,

# Standardizing the features
scaler = StandardScaler()
X_test_scaled_late = scaler.fit_transform(all_features)

y_test_binary = np.where(y_test_combined[:, 0] == 'high_stressed', 1, 0)
y_test_binary = y_test_binary[:min_samples]

# Initialize late fusion model
late_fusion_model = LogisticRegression()

late_fusion_model.fit(X_test_scaled_late, y_test_binary)

# Obtain predictions from the late fusion model
late_fusion_predictions_proba = late_fusion_model.predict_proba(X_test_scaled_late)[:, 1]

# Adjust the decision threshold based on precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test_binary, late_fusion_predictions_proba)

# Finding the threshold that maximizes F1-score or any other preferred metric
best_threshold = thresholds[np.argmax(2 * precision * recall / (precision + recall))]
late_fusion_predictions = (late_fusion_predictions_proba >= best_threshold).astype(int)
```

Figure 23: Late Fusion model implementation.

```python
# Evaluating performance
print("Binary Classification Model Performance:")
print(classification_report(y_test_binary, late_fusion_predictions))
print("Confusion Matrix:")
print(confusion_matrix(y_test_binary, late_fusion_predictions))
```

```
9/9 [==============================] - 0s 4ms/step
175/175 [==============================] - 70s 383ms/step
Binary Classification Model Performance:
              precision    recall  f1-score   support

           0       0.86      0.75      0.80       159
           1       0.73      0.85      0.79       129

    accuracy                           0.80       288
   macro avg       0.80      0.80      0.79       288
weighted avg       0.80      0.80      0.80       288

Confusion Matrix:
[[119  40]
 [ 19 110]]
```

Figure 24: Classification report of late fusion model.